Aalto University
School of Business

User-Centered Design: A Source of Sustainable Competitiveness for Software SMEs

Bachelor's Thesis
Joel Novamo
26.08.2017
Business Technology

Approved in the Department of Information and Service Economy xx.xx.20xx and awarded the grade

Aalto University
School of Business

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of bachelor's thesis

Author Joel Novamo

Title of thesis User-Centered Design: A Source of Sustainable Competitiveness for Software SMEs

Degree Bachelor's Degree

Degree programme Business Technology

Thesis advisor(s) Katri Kauppi

Year of approval 2017 **Number of pages** 26 **Language** English

Abstract

As technology continues to become a more prominent part of our daily lives, the demand for different types of software systems continues to grow as well. Although discussion about the rise of technology often revolves around large, globalized companies like Google, Microsoft, and Amazon, the majority of the businesses answering to the rising demand for software are small and medium sized enterprises(SMEs). These companies practice agile software development methods to develop software with smaller development teams to compete in the software market. The competitiveness of these companies is hindered however by the multitude of natural limitations that come with their smaller size, such as low financial capacities, and lower levels of overall expertise. The purpose of this thesis was to research the relationship between software SMEs and user-centered software design, and to arrive at a practical conclusion about how software SMEs could use this design method to sustain their competitiveness.

The thesis first provided a literature review about the different limitations and advantages that are unique to software SMEs. The main limitations discussed in the review were a lacking ability to finance endeavors that failed to bring in immediate returns and the inability for developers to rely on the development practices found in literature. The review also discussed the strengths of being able to work in a smaller cohesive unit, and having an improved flow of ideas due to a more flexible company hierarchy.

The literature review continued by defining and discussing the properties of user-centered design. This section of the literature review also went into subsections of discussing different aspects of the user-centered design process, such as its emphasis on contextual understanding, what makes it an iterative process, and usability testing. The section also reviewed the concept of discount usability engineering and the methods that it incorporates into the design process. The final part of the section introduces the concept of standardization as last resort method of design that larger companies are forced to use in their products.

Once the literature review has provided the reader with material about software SMEs and the user-centered design process, the thesis introduces and explains a new problem centric design methodology that utilizes discount usability engineering to keep costs down. The thesis concludes that this methodology can provide software SMEs with a source of sustainable competitive advantage.

Keywords User-centered design, SME, Usability, Usability engineering

Table of Contents

1. Introduction **4**

2. **Research Question** **4**

3. **Characteristics of a software SME** **5**

3.1 Financial deficit 5

3.2 Limited guidelines from literature 6

3.3 Strengths of software SMEs 7

4. User-centered design **8**

4.1 Contextual understanding of the user 9

4.2 An iterative process 10

4.3 Usability 12

4.4 Discount usability engineering 14

4.5 Standardization 15

5. Methodology proposal **17**

6. Conclusion **23**

6.1 Future research 23

1. Introduction

As the degree to which technology is incorporated into business continues to grow, the demand for specific types of software continues to increase as well. A major portion of the companies supplying this type of software is made up of small and medium sized enterprises(SMEs) that work with cohesive teams and innovation driven ideas to try and maintain their own niche in the software market (Almomani, Basri, Mahamad, Bajeh, 2014). For example, about half the software companies in Canada and Chile have less than 10 employees, and in the Unites States the portion is as high as 78% (Quispe, Marques, Silvestre, Ochoa, Robbes 2010). In order to maintain a competitive advantage, these types of companies need to work around obvious limitations such as small numbers of personnel, narrow profit margins, and tight schedules (Almomani et al. 2014). Working with these limitations can make it difficult to practice ideal software development methods such as extensive requirements engineering, thorough system modeling, and process management (Eito-Brun & Sicilia 2016). Without the ability to practice these types of ideal software development practices, SMEs need to draw on their strengths to find other ways of maintaining a competitive advantage. The objective of this thesis is to draw on ideas in related literature to provide an argument for how software SMEs can use customer centered software design to create and maintain a competitive advantage.

2. Research Question

Q: How can small and medium sized software development companies leverage user-centered software design to sustain a competitive advantage?

The original research question was changed during the research process away from measuring the extent to which user-centered design could help software SMEs in order to bring practicality and real usefulness to the reader. The objective of this study is to provide an analysis that leads to a conclusion with practical value and that an owner of a software SME can bring into actual use in a company.

The purpose of this thesis will be reached through a literature review, though which relevant topics will be introduced and discussed. The first part of the thesis discusses the shortcomings of software SMEs that will be needed to be worked around when suggesting a software development methodology. The second part of the thesis will consist of a brief explanation of the strengths that this

methodology will be utilizing. The third part of the thesis will explain user-centered software design with referencing to specific techniques and what they would look like in an SME. This part will continue by discussing Jakob Nielsen's Discount Usability Engineering and how it plays to the strengths discussed in the previous passage. The final part of the essay will provide a conclusive argument as to how the ideas from the literature review support a certain customer centered methodology of software development.

3. Characteristics of a software SME

This section of the thesis will serve to provide a discussion about the characteristics of software SMES. The section will provide a general explanation of what counts as a software SME within the context of this thesis, as well as discussion about the limitations and advantages that are naturally a part of these companies.

The definition of an SME varies between countries. Most often the upper limit as to what can be counted as a medium enterprise is set at 250 employees, and 50 employees for a small enterprise ("Small and medium", 2005). These limits are used for legal definitions within the EU ("Small and medium", 2005). However, in the United States the upper limit of an SME is up at 500 employees (King, 2015). This shows that the exact definition of what counts as an SME is somewhat up for interpretation. Since the purpose of this thesis is to create guidelines for companies that need to work around limitations that are being caused directly by their size, the definition of an SME within the context of this thesis will use the upper limit of 250 employees. This also means that the relevance of the topic being discussed in this thesis will increase as the severity of the limitations within the target company increases. Hence, as we go on to talk about the limitations that software SMEs are faced with, each company needs to measure the applicability of the topic based on the level to which these limitations relate to their company.

3.1 Financial deficit

One of the biggest limitations that software SMEs face is the lack of financial capacity to fund anything that does not bring in immediate returns. According to Eito-Brun and Sicilia (2016) the inability to invest in research and development is a major reason for why small companies struggle to practice systematic innovation in order to improve their products and services. Without the ability to bring in new ideas and methods into the company's practices, a software SME will have a hard time competing if a larger company chooses to enter its niche to provide a similar product or service. A

larger company that is able to invest in quality and process management will be able to eventually offer the product or service either at a higher insured quality or at a lower price, meaning that the SME needs to find value from something else in order sustain a competitive advantage. Eito-Brun and Sicilia (2016) bring up an interesting point in describing innovation in software development as something that comes naturally as developers find ways to meet the user requirements of the system. They say that the user requirements provide a problem to be solved for the developers, and sometimes innovative ideas with practical use in other related projects are used to solve the problems. However, they continue by saying that the developers at software SMEs don't have the luxury to look into the flexibility of their ideas and so innovation is forgotten once it gets implemented in a single system (Quispe et al. 2010). This would support that the financial limitation that prevents software SMEs from managing their developments processes is also limiting the companies from finding new innovations from which to leverage advantages.

In addition to hindering innovation management, a software SME's low financial capacity affects the overall expertise that can be expected from its employees (Quispe et al. 2010). Due to the complexity of software the level of skill of an employee can vary greatly, and higher levels of programming ability and other related expertise lead to higher pay-checks (Naalisvaara, 2015). Software SMEs will naturally be inclined to hire personnel who are willing to work for smaller amounts of pay. This is again following the ideology of SMEs looking for short term gains from their investments, but overall lower levels of expertise will bring inefficiency to the development process and lower levels of quality to the finished product. In the long-run the lower level of expertise will hurt the company's ability to meet user expectations and compete within its market.

3.2 Limited guidelines from literature

Although software development as a process is a deeply studied topic with lots of studies and literature to provide standards and guidelines for how it should be done, developers at small companies are still being left empty handed in terms of knowing how their specific company should be handling the process. This is largely due to the fact that the field of different activities that software development entails and the forms that the process can take is so varied that it's difficult to generalize what makes for a good software development process in a small company. Eito-Brun and Sicilia describe this by saying that "…the diversity in software technology is so wide that a single entity has difficulty

delivering comprehensive solutions…" (Eito-Brun & Sicilia, 2010, p.39), meaning that there doesn't exist one single solution that would comprehensively give answers to the entire field of software development in small companies. The theory behind software development has been laid out in books such as Ian Sommerville's *Software Engineering* (2016) text book, but the theory applies largely to bigger companies that have an easier time generalizing the process and that can spare resources to practice extra development steps in order to bring long-term sustainability to the company. Quispe et al. (2010) support the existence of this problem by saying that the developers at small companies are fully aware of the benefits that ideal software development practices can bring, but aren't able to agree on whether or not these practices have merit in the context of small companies. This is a serious limitation for software SMEs as a large portion of the expertise that is being brought into the company is being wasted as developers aren't able to practice software development the way they know to be ideal. Furthermore, it's unlikely that developers at these types of companies will be able to find their own ideal methodology in the context of their company as a tight schedule and limited financial capacity are sure to discourage experimentation.

3.3 Strengths of software SMEs

As the purpose of this thesis is to propose ways through which software SMEs can leverage an advantage in their market, it's important that we understand the advantages on which these SMEs can rely. Naalisvaara (2015) states that one of the advantages that small software companies benefit from is a flexible organizational structure as a result of lower numbers of personnel in managing level positions. According to Sommerville (2016), with a looser hierarchy in the workforce the company's employees have an easier time communicating their ideas, and that a better flow of ideas between relevant individuals will lead to a more flexible development process. This flexibility gets enhanced by the fact that these small companies often have development teams that can all fit into a single conference room, meaning that the entirety of the team gets the same information passed to them, meaning that time doesn't get wasted and there is less chance of problems being caused by misinformation (Quispe et al. 2010). A flexible structure also allows for all the personnel in the team to be continuously up to date on the big picture of the current project, instead of having to focus only on that tasks that have been passed down by management. The complexity of software development makes this an incredibly important advantage, as having a team of developers and designers who all

know exactly what is required of the system greatly enhances the working efficiency of each individual in the team, as well as bringing more structure to the software architecture (Sommerville, 2016).

In addition, the use of a small development team that can work the development process as a tight unit works right into the strengths of modern agile development techniques. Sommerville (2016) defines agile software development methods as methods that develop the software in increments and that stress informal communication and minimized documentation. According to Sommerville (2016), agile software development methods were originally developed for use within small programming teams and for developing small and medium systems. Software SMEs can use this to their advantage as being able to brief the entire development team about the current increment and being able to react to problems that are encountered only once development has started allows SMEs to maximize the benefits of agile software development.

4. User-centered design

This section of the thesis will provide a literature review about the topic of user-centered design. The concept as a whole will be defined and explained, and an explanation and discussion of its different concepts will be provided. The concepts will also be discussed from the perspective of a software SME in order to show their relevance to the core topic of the thesis.

As the name would suggest, *User-centered software design* is a design process that centers its focus around letting the user influence the design of the system (Abras, Maloney-Krichmar, Chadia, 2004). Abras et al. (2004) explain that the term covers a wide range of different methods that can be used to actually involve the user, but that the primary factor is that the user is in some way influencing the design. Norman (2013) uses the term human-centered design to discuss the same topic. He defines the process to be one that "…puts human needs, capabilities, and behavior first, then designs to accommodate those needs, capabilities, and ways of behaving" (Norman, 2013, p.8). Norman goes on to say that this type of design process needs to start from a solid understanding of the needs of the people that will be using the product (Norman 2013). This section of the thesis will provide discussion of different aspects of user-centered software design, and explain how these aspects make it a source of competitive advantage for software SMEs.

4.1 Contextual understanding of the user

For software developers to be able to design a system that supports the special needs of the current customer, they need to come to understand what it is exactly that the users of the software are going to be doing. For this to be possible, developers need to study the context that the users are coming from and make relevant observations about critical elements of the user's work environment. For this, developers can use *contextual inquiry interviews*, one-to-one meetings with users where developers go to visit the workplace of the user to observe the work that the system is going to be used for (Naalisvaara, 2015). During the interview, the developer tries to make as many relevant observations as he can about the critical processes in the user's work environment by both asking questions from the user as well as observing the user's work in progress (Naalisvaara, 2015). Nielsen (1993) states that interviewers should lean towards doing as little as possible to interfere with the work being observed, saying that the goal of the interviewer is to be virtually invisible for the user. Being able to observe the user doing actual work is important as users typically aren't aware of all their small habits and so would be unable to provide complete information through just an interview (Naalisvaara, 2015). Once observations have been made, the developer and the user discuss the observations so that the developer can get more insight as to the thought processes of the user (Holtzblatt, Wendell, Wood, 2005). Understanding the user's thought processes can help the development team get a better understanding of what it is that the user really needs from the system, including the amount of variation that needs to be available in each task. Furthermore, understanding the user through contextual observations can lead to value being added to the new system, as developers may be able to diagnose what needs to be fixed in the current work processes. However, Naalisvaara (2015) states that inquiry interviews suffer from the drawback that their accuracy and usefulness are measured by the skill of the interviewer. If the developer that is sent to do the interview has poor analytical skills or isn't good at reading people, the quality of the qualitative data will suffer.

In order to connect the contextual understanding brought by interviewing the user to a software design the development team can use *work modelling* to compile the relevant insights into an intuitive model that brings structure to the critical processes that need to be supported by the software system (Naalisvaara 2015). A work model is able to bring qualitative data into a form that developers have an easier time translating into system requirements (Naalisvaara 2015). Work models are extremely useful for user-centered software design as they bridge the gap between qualitative requirements and system

specifications. Furthermore, simplifying the big picture of what the system is able to do helps bring in the customer to review the design of the system and give input as to whether or not they think their needs are being met by the design.

From the point of view of a software SME, getting to a point of complete comprehension of the user and its needs is extremely valuable because it gives the entire development team a clear image of what the system being developed needs to become. As discussed earlier, software SMEs have the advantage of being able to work in a flexible work environment where the entire development team is able to work together on a single project under a loose hierarchy. Ideas in this sort of work environment are free-flowing and developers are able to easily communicate with one another. This means that once one person in the team is able to make a critical observation about the user and has an idea about how that observation should affect the system design, the idea is easily transmitted to the rest of the team without need for formal documentation. Once the idea is brought to the entire development team, the task of implementing it in the next iteration of the system becomes easier, and the implementation process becomes more likely to succeed since the team as a whole is likely to be on the same page about how the new idea should be implemented in the grand scheme of the system. All of this would prove much more difficult in a larger organization, since the information about a new implementation would need to travel through more people and through stiffer communication channels (Sommerville 2016).

A challenge that SMEs need to be able to deal with when trying to analyze the needs of the user is that gathering information about the user and creating usable models at proper levels of sophistication can be extremely time consuming and can lead to additional overhead for the company (Naalisvaara 2015). As was discussed earlier, SMEs typically operate under stricter budgets and tighter schedules, which means that having to spend time and money on user analysis is relatively larger of an investment for an SME as it would be for a larger company. The process of interviewing and modeling would need to be managed to fit within the SMEs limits, which would prove difficult for a small company that doesn't have the time or resources to spend on process management.

4.2 An iterative process

Due to the nature of agile software development and the difficulty of designing a system architecture that fits the needs of the user, the user-centered design process shouldn't be done at once but by an iterative process that works towards the final design (Abras et al. 2004). An iterative process

works to take the primary evaluation of the user's needs and update it according to changing requirements, observations, and solutions that come with each iteration (Abras et al. 2004). Iterative software development allows for the company's development team to learn about the system and what it needs to look like as the system is built. In addition, developers will be able to look for feedback on each iteration of the system so that required changes will be limited to the area that was most previously completed. From a user-centered design view point this is incredibly important as the development team is unlikely to fully understand exactly what the user needs the system to look like at the beginning of development. Being able to test single iterations of the system and learning from said testing will lead to a system that has had time to mold itself into what the user really needs.

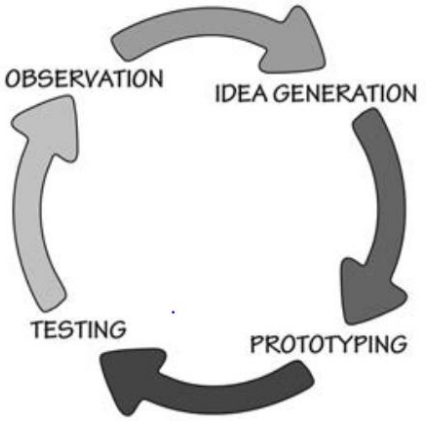

Figure 1: The Iterative Cycle of Human-Centered Design (Norman 2013, pg. 222)

Figure 1 on the left is from Donald Norman's 2013 edition of *The Design of Everyday Things* and it serves to illustrate the phases of the iterative user-centered design process. The cycle is also called the *spiral method* to emphasize that each cycle progresses the product's development. The process starts with the *observation* phase in which designers make their initial analysis of the context of the user. Both Nielsen (1993) and Norman (2013) emphasize the importance of getting to observe the users in their natural, undisrupted work environment in order to avoid flawed data. The process then moves on to *idea generation* during which designers discuss their observations and come up with ideas for how the observed activities could be fulfilled. The third phase of the process is *prototyping* where the team attempts to build mock-ups of the ideas from the previous phase in order to do initial testing of separate parts of the system. Norman (2013) describes there to be a multitude of ways that this prototyping can be carried out, such as pencil sketches, cardboard models, and images made with drawing tools. The overall goal of this phase is to find the truth about the practicality of the ideas before they are added to the product. The final phase of each cycle is *testing* which obviously includes having target users test the system while under observation. The chosen users should represent the target users as closely as possible, and the activities done during the test should be as close as possible to the activities that will be done with the system in practice. Once testing is complete, the cycle returns to the observations phase, where designers make observations based on the results of the tests. The

designers make decisions on what worked and did not work in the most previous iteration, and use these observations to come up with new ideas once again. The iterative process continues until the product reaches a level that the developers find satisfactory (Norman, 2013).

According to Sommerville (2016), the focus on iterative development brought by agile software development methods brings the benefit of allowing the development team to react to changing customer requirements. Customers are prone to change their initial requirements for the system once they see early versions of its design, which means that if a company wants to allow the user to influence the final product, it needs to be able to react to these changing requirements. Being able to receive feedback from each iteration of the system allows developers to make changes to the system, and hence allows the company to create a product that is shaped around what the user asked for.

From an SME's stand-point the ability to manage the system's development iteratively is advantageous as the company will be able to quickly adapt to changes being dictated by the customer. Also, being able to check in with users with each iteration to look for possible problems allows the company to limit the amount of time that needs to be spent on fixing newly risen issues, as the developers will always know exactly what it is that caused the problem. Furthermore, splitting the development process into iterations makes it easier for the company to invite the customer to be a part of the development process. If the process were to be done with one iteration, a customer would have a hard time understanding what was going on in the middle of development as there would be no checkpoints or guidelines for what the system currently looked like. The customer would simply be left to wait for the finished product which would then need to be completely renovated if the customer introduced required changes. With the iterative process, the customer can take part in meetings about newly developed iterations and so can keep tabs on what the system currently looks like. Not only will this bring a sense of co-creation to the process, but it will also improve communication and the speed at which requirements changes can be introduced to the development team. All of this would lead to an overall better development process within the company, as well as a high-quality user experience for the customer.

4.3 Usability

The methodology of taking the needs of the customer into account during the design process is clearly common in any form of business, but from a software development standpoint the methodology gets more specific, as the product being created for the customer needs to be able to both fit the specific

needs of the customer as well as meet the required usability standards of the customer so that the system can actually be of use. Nielsen (1993) states that usability shouldn't be thought of as a single decisive property but as multiple factors that together measure the usability of the system. Nielsen (1993) divides these factors into five usability attributes: Learnability, Efficiency, Memorability, Errors, and Satisfaction. In order to create a system design that meets the requirements sourced from the user, developers and designers need to be able to find a design that fills these attributes from the point of view of the user. An iterative design process can help developers test the current design according to these usability attributes. For example, the newest iteration of the system might have added a new possibility for customization within the software, and the developers would be able to utilize user-tests to see how well users are able to grasp the mechanics of the new addition. The level of external assistance required by the users would tell the developers whether or not the system possesses a sufficient level of learnability.

A straight forward method for testing the usability of the system during a user-centered design process would be prototyping, which includes recruiting chosen users to try using the current version of the system while under observation (Naalisvaara 2015). From a user-centered design standpoint this method of testing is excellent as it brings in actual users to test the system, providing the developers with direct feedback from actual users and bringing the customer into the design process. Abras et al. (2004) state that the prototyping process should be separated into three different sections. The first section includes basic conceptual modeling where mock up versions of the system are analyzed and evaluated with the user. This section is done before any coding is begun. The second section consists of using prototypes of early coded version of the system that can provide developers with fast feedback about the usability of the system. An iterative process is again used to test different version of the code in order to refine its usability attributes. Finally, the third section consists of testing the system once the user interface is complete and the final iteration can be compared to the usability goals set for the system at the beginning of the process (Abras et al. 2004).

Another form of testing the usability of the system is through *heuristic evaluation* (Naalisvaara 2015). This testing method is especially attractive from the point of view of a smaller software company as it allows for developers to test the usability of an iteration without investing in time and money to organize a formal testing session with users. This is because the heuristic evaluation process has the developers of the system systematically test its usability according to set usability guidelines

(Nielsen 1993). This means that since the process is based off set guidelines, even smaller companies that are unlikely to employ usability experts can utilize the process. Nielsen (1993) states the overall goal of heuristic evaluation to be finding usability problems in the user interface so that they can be solved in the next iteration of the system. However, Nielsen (1993) continues by saying that the formal usability guidelines reach a high level of complexity, and that this causes this type of evaluation to seem intimidating to some developers. Nielsen (1993) believes however that a heuristic evaluation can be done to some extent through common sense alone, which would mean that this method can prove advantageous even to development teams with lower levels of expertise.

4.4 Discount usability engineering

Usability testing is something that seems like the last thing that a strictly budgeted SME would be able to invest in. Laboratories meant specifically for usability testing are known to be extremely expensive, and hiring usability experts is not within a typical SME's budget (Abras et al. 2004). However, in 1989 Jakob Nielsen published a paper titled "Usability Engineering at a Discount", which introduced the topic of *Discount Usability Engineering*. Discount Usability Engineering is a software engineering methodology that revolves around testing the usability of a system while avoiding all the methods that have high price tags attached. The 1989 paper named 3 methods that allowed for this type of usability testing while keeping costs low. The first was *simplified user testing* which included basic user testing with a very small number of testers and a focus on qualitative observation and the thinking-aloud method. The cost of this type of testing stays relatively low as only a minimal number of testers are required, and the testing is done with open communication and an informal setting which means that there is no need for high investments. The second method was to use *paper prototypes* to map single paths through the theoretical user interface. These types of prototypes could be created quickly, cheaply and led to quick discussion and feedback from users. The main idea behind these prototypes was to single out individual logical paths within the design to avoid having to deal with the complexity of the system as a whole. The third method was heuristic evaluation, which is easy to understand as it allows for low cost and low expertise usability testing (Nielsen, 2009).

In his book *Usability Engineering* (1993), Nielsen updated the list of methods by splitting user testing into *User and task observation* and *Simplified thinking aloud*, and using the term *Scenarios* to describe extremely simplified prototypes that only show the user interface portion of the system. Nielsen (1993) goes on to specify that the heuristic evaluation method should be limited to 10 rules that

rely on testing the system in a broader sense. Regardless of the specific terms used however, the overall idea behind Nielsen's Discount Usability Engineering methodology is the same, in that it includes any and all methods of testing the usability of a system while keeping the costs to a minimum. According to Nielsen (1993), companies should overlook the suggestion that the methods listed under this methodology are cheap in price due to their ineffectiveness, saying that companies can gain similar benefits from cheaper methods as they would from more carefully crafted and expensive counterparts.

Nielsen (1993) emphasizes the point that Discount Usability Engineering is meant to be a method of testing that aims to find most of the problems in the system instead of all of them. He agrees with the argument that more careful methods would produce better results and hence that a discount methodology is not perfect, but argues that the high costs and higher requirements of expertise that these methods would require make the discount methodology a more viable choice for software designers.

In the year 2000 Patricia Yao and Paul Gorman tested the effectiveness of Discount Usability Engineering by applying it to web-based medical knowledge software. The test involved professional physicians being asked to do different tasks on two different versions of the system. The second system was the redesign of the first system that had been engineered using discount methods. The test recorded the success rate and time of completion for each task and the results were averaged to provide a comparison of usability between the two versions. The test showed that the first version of the system had an overall average success rate of 58.36% while the second version showed a 76.55% success rate. Furthermore, the first version of the system had an overall average completion time of 40 seconds while the second version's average time was 34.6 seconds. These results show that just one run of discount usability testing showed a massive increase in overall usability in the system. The test did also show that in some cases the new system performed worse, but in the context of an iterative process these cases could easily be corrected back to their better versions (Yao & Gorman, 2000).

4.5 Standardization

Standardization when designing a product is the implementation of a standard function within the product that has to be learned by the user, meaning that the function does not adhere to the common practices of the user but instead introduces a new type of functionality (Norman, 2013). Norman (2013) explains standardization as the last resort in a user-centered design process, and that standardization should only be used once "…no other solution appears possible" (Norman, 2013, p.155). In other

words, standardizing the design should only happen once it is deemed that it is too difficult to mold the design to fit within the boundaries provided by the user, and that standardization is basically giving up on meeting the user's needs and moving to the best alternative. This explanation of standardization describes the way large companies deal out software. Software developed by large companies is meant to bring in larger amounts of revenue, which means that the software must be able to be applied to use for all the company's customers. Although due to higher investments in process management bigger companies are able to offer a higher quality standard with their products, the software is never fully customized to any specific user. The software is targeted at a wide variety of different types of users that all share some common theme that has them buying this software in particular. Norman (2013) uses the example of how the way people tell time is standardized to being two cycles of 12 hours rather than just a single cycle of 24 hours. Another example that Norman (2013) uses is the way that all clocks are designed to rotate clockwise, pointing out how difficult it would be to tell time if a clock were to be designed to rotate counterclockwise. There is no special reason for why the hands have to rotate clockwise, but that is the way that clocks have bene standardized and so that is how people have learned to tell time. The standard way is in no way logical or natural but it is accepted simply because it has been set as the standard.

A similar parallel could be drawn for people who install Microsoft Windows onto their new home computer without even thinking about other alternatives. Windows as an operating system is not molded towards any certain user, but it's a standardized system that allows any user to use their computer because the way windows works is such an accepted standard that anybody that knows how to use a computer also knows how to use Windows. However, Windows had to be taught to everybody at some point. Norman (2013) describes the need to be taught how to use something to be a key attribute in a standardized product. Due to standardization Microsoft is able to sell its operating system to anybody that wants to buy a computer (unless that customer chooses a competitive product that follows the same basic guidelines that Windows does). However, the key in Norman's (2013) description of standardization is that it is the last resort in user-centered design, and that being able to avoid it will lead to a better product.

5. Methodology proposal

This section of the thesis will draw on the material provided in the literature review to provide an answer to the research question. The answer will be discussed and its general framework will be presented.

Through the course of this thesis so far we have been able to review the characteristics of software SMEs, including the limitations and advantages that they are generally working with. In addition we have reviewed numerous different aspects of user-centered design to see what types of methods it utilizes, and how those methods might look within the context of an SME. In order to reach the objective of this thesis and to answer the original research question, we needed to come to a conclusion about how the different aspects of user-centered design can be utilized by software SMEs to compete effectively while keeping in mind the limitations and advantages that were discussed. The methodology we are proposing is:

A problem based approach to user-centered software design that utilizes Discount Usability Engineering

The main reasoning for this methodology comes from the need for SMEs to be able to work around their limitations while making use of their advantages. As was discussed in section 3.1, SMEs suffer from the lack of ability to make large investments into endeavors that do not bring in immediate returns. This means that any new methodology would be useless in practice if it required the implementation of business practices with high costs. In the case of usability testing this would be especially problematic as traditional testing includes expensive testing facilities and high levels of usability expertise. Implementing Discount Usability Engineering practices into the design and testing process works around this hindrance as the method is obviously centered around the very purpose of keeping both the required levels of expertise and investment low.

In addition to keeping costs low, discount usability testing methods emphasize the use of qualitative data and communication between human beings. The emphasis on qualitative data means that the improvements between each iteration are made due to logical observation rather than anything that software developers would need to read from literature. This in turn means that developers would not need to rely on ideal software development practices to bring improvements to the system as they would be able to rely on their own critical thinking and group discussion to implement improvements between iterations of the system. This property of discount usability testing methods further helps

developers at SMEs work around the fact that they are not able to find exact guidelines for their work from literature.

The problem based approach of our methodology is sourced from Norman's (2013) definition of what it means to design based on the user. Norman (2013) states that the most important factor of coming up with a design is to make sure that the design works as a solution to the right problem. He goes as far as to say that the quality of the design is meaningless if this right problem is not being solved (Norman 2013). Within the context of software development, this means that the quality of the software being developed has little meaning if the system's design is missing out on the fundamental needs of the user. According to this definition and emphasis on being able to understand what the core problems of the user are, we believe that software SMEs can use their better ability to come to a full contextual understanding of the user in order to bring better products to the market and hence sustain a competitive advantage. Compared to larger software companies that have to practice standardization to make their products fit into the use of large numbers of customers, software SMEs have the luxury of focusing on smaller projects and individual customers. This means that SMEs can practice software development that draws its basis from understanding the underlying problems that its targeted users are having. A small development team that works in a co-creative manner with the customer can have an easier time studying observations made about the targeted users, and drawing conclusions about what it is that the customer really needs out of the product. By being able to do this, SMEs can produce products that are better molded for the specific needs of their customers, and create value for their users that larger companies won't be able to compete it. The framework of the problem centric methodology is represented by figure 2 below.

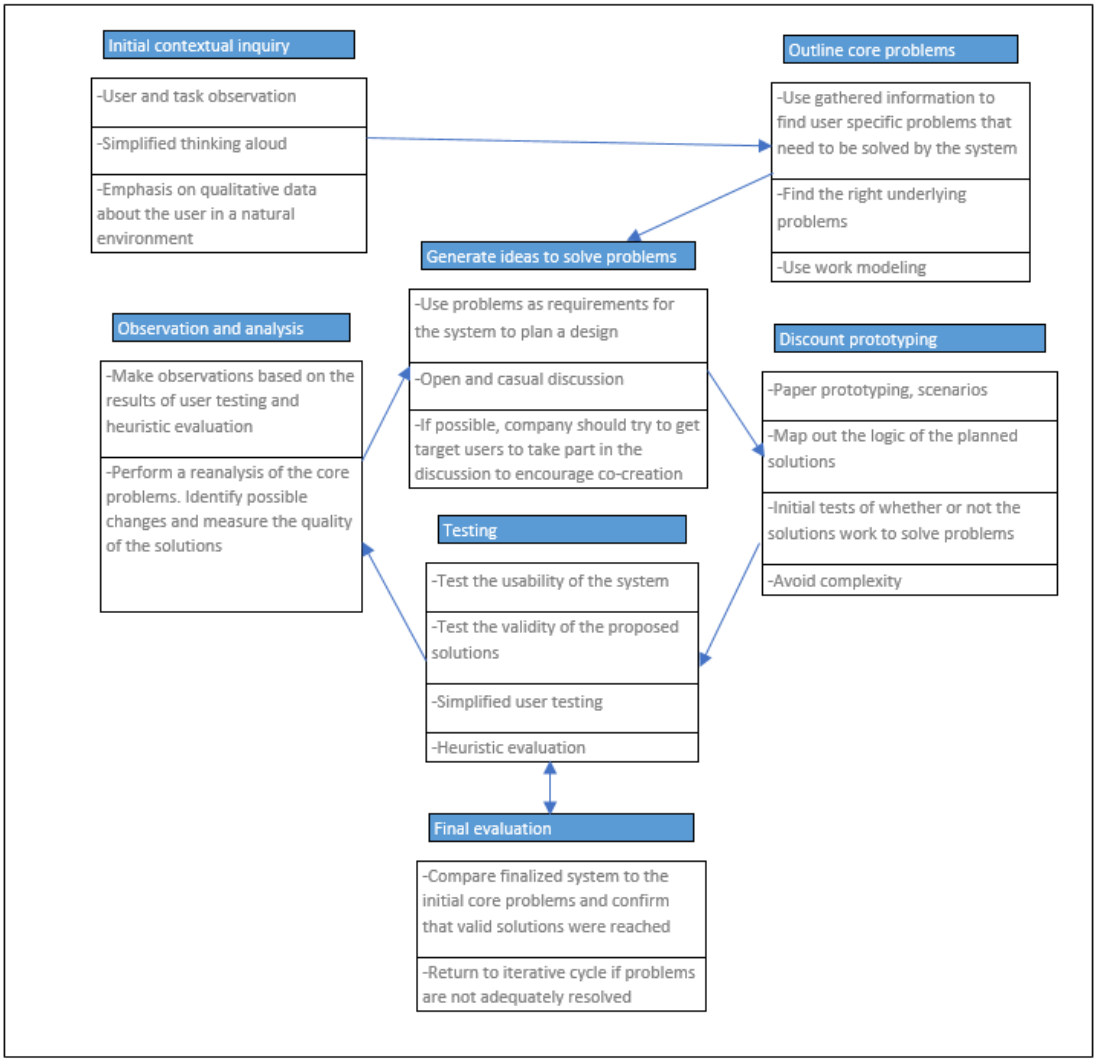

Figure 2: Framework of problem based approach to user-centered software development that applies discount usability engineering

The framework presented in figure 2 serves as a guide for how software SMEs can approach the problem based approach to user-centered software design. The purpose of the model is not to set strict rules that companies have to follow in order to find success using this methodology, but to provide a basic outline of how a company can incorporate the iterative spiral method, contextual understanding of the user, and discount usability engineering methods into a complete development process. The model also has the purpose of providing SMEs with a visual representation of the methodology in order to encourage its incorporation into actual business practices, and hence bring practical value to the proposal of this thesis.

The framework in figure 2 shows that the design process starts with an initial contextual inquiry. This part of the process is done through face-to-face interaction with target users during contextual inquiry interviews as well as silent observing of the user. The purpose of the contextual inquiry is to gather as much relevant qualitative data about the user's work as possible. More importantly than volume however, the quality of the data needs to be high as the design process relies on these qualitative observations to begin with a clear picture of what is required by the user. In order to gain a full understanding of the processes in the user's work, interviews should utilize the thinking aloud method. Being able to hear the steps of the process that the user is doing will help the interviewer make more accurate observations about the processes being done, which will lead to a more solid contextual understanding of the user's requirements.

The model goes on to an analysis phase where the design team studies the observations made about the user and tries to interpret the core problems that need to be addressed by the system design. Although the nature of the iterative design process helps in correcting errors made during this phase, it is still important that the design team tries to get as close to the right problems as possible during this phase in order to make the design process more efficient. In practice, being able to arrive at the right problems during this phase will reduce the number of iterations that will be required to complete the project, and hence save the company valuable time and funds. By the time this phase has ended, the entire development team should be on the same page both in terms of understanding the context of the user as well as the core problems that the system design needs to be pointed toward solving. The ability of SMEs to communicate effectively should enhance the company's ability to arrive at an agreed upon conclusion effectively, which will help keep the design team on the same page about the form that the system design needs to take.

Once the design team has identified the core problems around which to design the system, the process enters the iterative cycle that was discussed in section 4.2. Within our methodology, the cycle is executed with the use of discount usability engineering methods. Figure 2 shows when and how these methods are to be implemented. The cycle starts with the idea generation phase where the design team brainstorms possible solutions for the core problems. The plans for the system design should be generated up from these problems in order to ensure that the final system design ends up being one that contains solutions for these problems. Development teams at software SMEs should utilize their ability to carry out free flowing and casual communication to allow ideas to flow freely during this phase. The

best case scenario in this phase would be to get a target user of the system to take part in this process in order to add a voice with hands-on experience of the work that the system will be used for. This would provide an early measurement of the practicality of early ideas, enhancing the effectiveness of the discussion and speeding up the process.

After discussing possible ideas for a system design, the process moved on to the prototyping phase during which the ideas are implemented using paper prototypes and scenarios. The prototypes should be kept extremely simplistic for two reasons. The first reason is that simple prototypes will be easy to make in terms of time and money which will keep the overall cost down. The less is invested into the creation of the prototypes the more effective the discount method becomes. The second reason is to avoid dealing with too much complexity when discussing the basic work processes that are being fulfilled by the system. As has been emphasized, the end goal of the proposed methodology is to develop software based on the core problems of the user. This means that the development team cannot afford to lose sight of the principle solutions that need to be a part of the system. Dealing with overly complex prototypes has the risk of creating ambiguity in the functionality of the system, which could cause the system to end up with a design that does not fulfill its initial intended purpose. The prototypes should simply be able to test the logic of the new solutions before they are implemented and further tested.

Once the best new ideas have been implemented into a working prototype, the process moves on to testing. The testing phase needs to provide data on two factors of the system. The first is the system's usability. The system's usability can be tested through either heuristic evaluation or through simplified user testing. Once again, the process should maximize the effectiveness of the discount usability engineering methods in order to bring high quality results while keeping costs down. User testing should be executed with very small numbers of users, and with an emphasis on qualitative observations and asking the users to explain their thought processes. Heuristic evaluation should be kept at a simple level with Nielsen's (1993) recommended 10 rules as this will keep the required expertise for the evaluation low. Whether to test using user tests or heuristic evaluation needs to be decided by the development team based on their own judgement of what is practical during the current iteration. For example, if users were found to struggle with a specific part of the system during the last user tests, more user testing will be required to see if the problem was solved with the solutions of the new iteration. However, if the system has been struggling with issues that were found through heuristic

evaluation, more evaluations will be needed to check if the problems persist. In addition, factors such as the availability of target users and time constraints can affect the decision of how testing should be executed. The second factor that needs to be tested for is the validity of the proposed solutions to the core problems. With each iteration, more functionality will be added to the system, which means that the system will continue becoming more complex. Testing needs to first check if the proposed solutions are solving these problems, and then make sure that the increase in complexity is not degrading solutions that have already been implemented.

The final phase of the iterative cycle contains observation and analysis of the test data. Overall this phase serves as an evaluation period for the solutions that were implemented during the last iteration. First the development team should evaluate the ability of the implemented solutions to solve the problems that they were intended for. Second, the team should evaluate the extent to which these solutions affected the usability of the system. On the first iteration, the emphasis of the analysis should stay on the core problems that were defined at the beginning of the process. After the first iteration, each analysis period should be a comparison to the last iteration so that both improvements and deteriorations can be tracked. It is important that the analysis notices deteriorations form the past iteration so that the system can be backed up to its original version if needed.

The development process continues the iterative cycle until the system reaches its final version. At this point, the process exits the cycle and enters the final evaluation phase as shown in figure 2. At this point the final version of the system is compared to the initial core problems that were identified at the beginning of the development process. The purpose of this phase of the process is to make sure that the system is still able to provide solutions to the right problems, and hence that the system is indeed meeting the specific requirement of the target user. If this evaluation fails, and it is discovered that the system is unable to provide the user with the correct functionality despite passing its tests, the system is sent back into the development cycle so that the discovered problem can be corrected. This final evaluation works as a safeguard in case the development team lost sight of the underlying needs of the user at some point during the development process.

The end goal of the methodology we have proposed and that is being represented by the model in figure 2 is to be allow a software SME to develop a product that fits the needs of the target user. We believe that using this methodology, SMEs can bring value to their customer despite their limitations in a way that larger companies would struggle to do. This argument is backed up by Norman's (2013)

explanation of standardization. Larger companies that have to introduce standardized functionality in their products will have a more difficult time reacting to the specific requirements of each customer the way SMEs are able to. This difference in ability to react to the core problems of the user is what we believe can give software SMEs the advantage in terms of competitive advantage in the software market.

6. Conclusion

The objective of this thesis was to provide the reader with an answer as to how software SMEs can concentrate on user-centered software development to sustain a competitive advantage in the software market. We discussed the limitations of SMEs that were needing to be worked around, along with some strengths that gave SMEs an advantage. After explaining the relevant topics of User-centered software design, we made the proposition that software SMEs use Norman's philosophy about user-centered design to build a software design process that revolves around finding the underlying problems that the users are facing, and that applies Nielsen's Discount Usability Engineering in order to work around the SMEs resource limitations. The argument for how this methodology allows for the SME to sustain a competitive advantage was also drawn from Norman's philosophy, which suggests that being able to mold the product according to the needs of the user is better than having to resort to standardization. We then presented and discussed the general framework of a development process that utilizes the proposed methodology, including how this framework implemented different discount usability engineering methods. The framework of the process shows how the core problems of the user can be solved and tested using discounted methods, and how this ultimately can lead to a user oriented software product. In the long run this will help software SMEs sustain a competitive advantage by allowing them to react to the specific characteristics of each user despite the limitations they are naturally faced with.

6.1 Future research

A concept that was left out of this thesis but that is interestingly tied to the topic is the concept of how innovation driven software development can be practiced in small software companies. This idea is discussed in Ricardo Eito-Brun and Miguel-Angel Sicilia's 2016 article *Innovation Driven Software Development: Leveraging Small Companies' Product Development Capabilities.* The reason

this concept is tied to the topic of this thesis is that both innovation and the methodology proposed in this thesis circulate around the creation of new ideas. Within our proposed framework, the development team needs to be able to come up with ideas for how to solve the core problems of the user. Likewise, Eito-Brun and Sicilia (2016) define innovation as transforming ideas into improved products. Furthermore, Eito-Brun and Sicilia (2016) introduce a model for how small software companies can practice innovation management despite the same restrictions that were discussed in this thesis. This would suggest that theoretically even a small company would be able to utilize the new ideas generated from our proposed problem centric approach to fuel the creation of innovative products or services. This topic of the relationship between our proposed methodology and innovation driven software development would be a possible topic for future research.

References

Abras, Chadia. Maloney-Krichmar, Diane, Preece, Jenny. 2004. *User-Centered Design.* W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications, p.1, 5, 8-9 [Accessed 31.7.2017]

Almomani, Malek. Basri, Shuib. Mahamad, Saipunidzam. Bajeh, Amos. 2014. *Software Process Improvement Initiatives in Small and Medium Firms: A Systematic Review*, 3rd International Conference on Advanced Computer Science Applications and Technologies, Department of Computer and Information Services, Universiti Teknologi Personas, p.162 [Accessed 31.7.2017]

Eito-Brun, Ricardo. Sicilia, Miguel-Angel. 2016. *Innovation Driven Software Development: Leveraging Small Companies' Product Development Capabilities.* IEEE Computer Society, p.38-44, [Accessed 31.7.2017]

Holtzblatt, Karen. Burns, Jessamyn. Wood, Shelley. 2005, *Rapid Contextual Design: A How-To Guide To Key Techniques For User-Centered Design*, Elsevier, Chapter 4, p.80 [Accessed 31.7.2017]

King, Tyler. 2015, *CRM Definition: What does "SMB" or "SME" mean?,* Less Annoying CRM. https://www.lessannoyingcrm.com/resources/smb_sme_definition [Accessed 24.8.2017]

Naalisvaara, Lauri. 2015. *Applying and Evaluating Design Techniques in a Small Company.* Aalto University School of Science Master's Thesis. p.17-21, 28 [Accessed 31.7.2017]

Nielsen, Jakob. 2009, *Discount Usability: 20 Years*, Nielsen Norman Group, https://www.nngroup.com/articles/discount-usability-20-years/ [Accessed 31.7.2017]

Nielsen, Jakob. 1993, *Usability Engineering,* Academic Press Inc. p. 17-20, 26-27, 155, 207-208, 225 [Accessed 31.7.2017]

Norman, Donald. 2013, *The Design of Everyday Things*, Basic Books, p.8, 155, 221-230, [Accessed 24.8.2017]

Quispe, Alcides. Marques, Maira. Silvestre, Luis. Ochoa, Sergio. Robbes, Romain. 2010, *Requirements Engineering Practices in Very Small Software Enterprises: A Diagnostic Study*, International Conference of the Chilean Computer Science Society. p.81-83 [Accessed 31.7.2017]

Small and Medium-sized Enterprises(SMEs), 2005, OECD Paris and Entrepreneurship Outlook. https://stats.oecd.org/glossary/detail.asp?ID=3123 [Accessed 24.8.2017]

Sommerville, Ian. 2016. *Software Engineering 10th Edition.* Pearson Education Limited. p.18, 45, 647, 663 [Accessed 31.7.2017]

Yao, Patricia. Gorman, Paul. 2000, *Discount Usability Engineering Applied to an Interface for Web-based Medical Knowledge Resource,* AMIA Symposium, Department of Medical Informatics and Outcome Research, p. 928-932 [Accessed 31.7.2017]